```

#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"

void error(char *name);
// initialize limitIntensity function
int limitIntensity(double value);

int main (int argc, char **argv)
{
    FILE *fp;
    struct TIFF_img input_img, color_img;

    if ( argc != 2 ) error( argv[0] );

    /* open image file */
    if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
    fprintf ( stderr, "cannot open file %s\n", argv[1] );
    exit ( 1 );
    }

    /* read image */
    if ( read_TIFF ( fp, &input_img ) ) {
    fprintf ( stderr, "error reading file %s\n", argv[1] );
    exit ( 1 );
    }

    /* close image file */
    fclose ( fp );

    /* check the type of image data */
    if ( input_img.TIFF_type != 'c' ) {
    fprintf ( stderr, "error:  image must be 24-bit color\n" );
    exit ( 1 );
    }

    /* set up structure for output color image */
    /* Note that the type is 'c' rather than 'g' */
    get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );

    // create 3-dimensional array storage_img via nested pointer memory 
      allocation
    double*** storage_img = (double ***)malloc(3 * sizeof(double **));
    for (int k = 0; k <= 2; k++) {
        storage_img[k] = (double**)malloc(input_img.height * sizeof
          (double));
```

```c
        for (int ht = 0; ht < input_img.height; ht++) {
            storage_img[k][ht] = (double*)malloc(input_img.width * sizeof(double));
        }
    }

    // initialize all indices of storage_img to zero
    for (int i = 0; i < input_img.height; i++) {
        for (int j = 0; j < input_img.width; j++) {
            for (int k = 0; k < 3; k++) {
                storage_img[k][i][j] = 0.0;
            }
        }
    }

    // nested for loop that covers each pixel
    for (int i = 0; i < input_img.height; i++) {
        for (int j = 0; j < input_img.width; j++) {
            // for each plane in RGB pixel
            for (int k = 0; k < 3; k++) {
                // assign to storage_img the term that will always exist (non-recursive component)
                storage_img[k][i][j] = 0.01 * input_img.color[k][i][j];
                if (i - 1 >= 0) {
                    // assign to storage_img the term that exists if i > 0 (recursive component)
                    storage_img[k][i][j] += 0.9 * storage_img[k][i - 1][j];
                }
                if (j - 1 >= 0) {
                    // assign to storage_img the term that exists if j > 0 (recursive component)
                    storage_img[k][i][j] += 0.9 * storage_img[k][i][j - 1];
                }
                if (i - 1 >= 0 && j - 1 >= 0) {
                    // assign to storage_img the term that exists if i > 0 & j > 0 (recursive component)
                    storage_img[k][i][j] += -0.81 * storage_img[k][i - 1][j - 1];
                }
                // populate output image method for color after calling limitIntensity function to ensure acceptable RGB values
                color_img.color[k][i][j] = limitIntensity(storage_img[k][i][j]);
            }
        }
    }

    /* open color image file */
    if ( ( fp = fopen ( "filtered.tif", "wb" ) ) == NULL ) {
        fprintf ( stderr, "cannot open file color.tif\n");
        exit ( 1 );
    }

    /* write color image */
    if ( write_TIFF ( fp, &color_img ) ) {
        fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
        exit ( 1 );
    }

    /* close color image file */
    fclose ( fp );

    /* de-allocate space which was used for the images */
    free_TIFF ( &(input_img) );
    free_TIFF ( &(color_img) );

    return(0);
}

void error(char *name)
{
    printf("usage:  %s  image.tiff \n\n",name);
    printf("this program reads in a 24-bit color TIFF image.\n");
    printf("It then horizontally filters the green component, adds noise, 
      \n");
    printf("and writes out the result as an 8-bit image\n");
    printf("with the name 'green.tiff'.\n");
    printf("It also generates an 8-bit color image,\n");
    printf("that swaps red and green components from the input image");
    exit(1);
}

// limitIntensity function definition
int limitIntensity(double inputValue) {
    // declare an integer variable newValue and initialize it to zero
    int newValue = 0;
    // if input value parameter is less than zero, assign new value to 0
    if (inputValue < 0) {
        newValue = 0;
    }
    // if input value parameter is greater than 255, assign new value to 
      255
    else if(inputValue > 255) {
        newValue = 255;
    }
    // otherwise, assign new value to the input value parameter re-cast as 
```

```
        an integer
    else {
        newValue = (int)inputValue;
    }
    return newValue;
}
```